Introduction to Python: Part I

Austin Godber
@godber
DesertPy Co-Organizer
Meetup, Github and http://desertpy.com (http://desertpy.com)

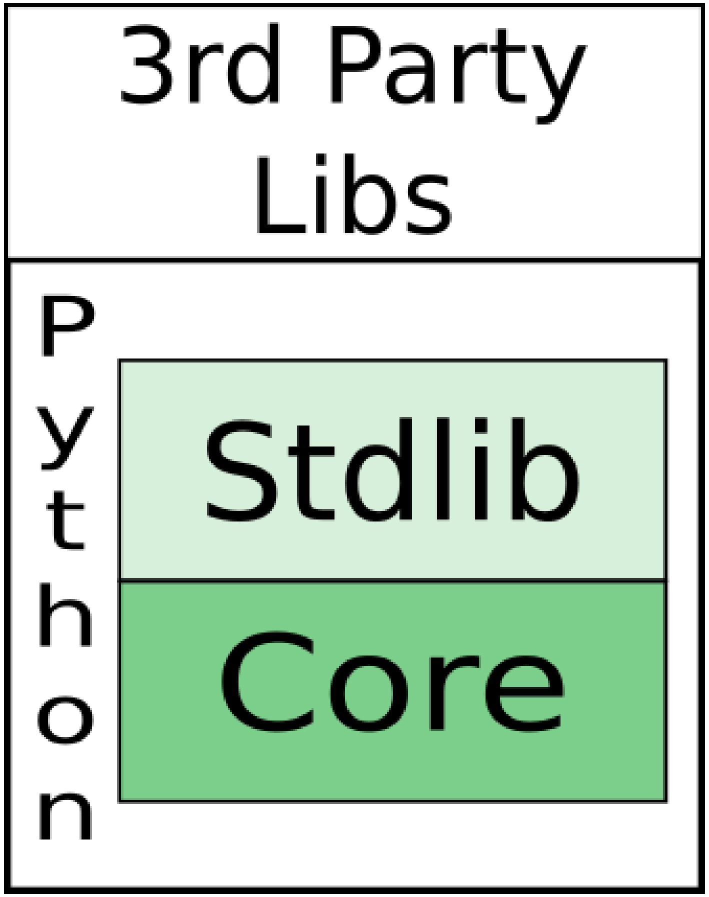
Desert Code Camp 2019 - 10/12/2019

DesertPy - The Phoenix Area Python Meetup Group

- Typically two meetups a month
  - 4th Wed night: Presentation meeting
  - 2nd Saturday AM: Open Hack
- Go to Meetup.com and search for DesertPy

DesertPy

Python Language



Python

- Python - https://docs.python.org/3/reference/index.html (https://docs.python.org/3/reference/index.html)
  - Python Interpreter: CPython, Pypy, IronPython(2.7), Jython(2.7)
  - Core Language Syntax, built-ins
- Python Standard Library - https://docs.python.org/3/library/index.html (https://docs.python.org/3/library/index.html)
  - C or Python modules included with Python
  - These modules need to be imported, e.g.: `import math`

3rd Party Libs

- 3rd Party Modules - https://pypi.python.org/pypi (https://pypi.python.org/pypi)
  - Install them with `pip`
  - These modules need to be imported
    - e.g.: `import requests`

Python 2 vs Python 3

Python 2 is EOL at the end of this year! So you should be using Python 3!

- https://docs.python.org/3/ (https://docs.python.org/3/)
- https://docs.python.org/2.7/ (https://docs.python.org/2.7/)

I'll be talking about Python 3.5+ today.

How to Start

- Install Python
- Get Text Editor
- Follow tutorial

Install Python

- Windows - Anaconda/miniconda, Enthought Canopy
- OS X - Default install, Homebrew, Anaconda
- Linux - Default install, Anaconda

Text Editor

- Editor
    - SublimeText 3 (https://www.sublimetext.com/)
    - Atom (https://atom.io/)
    - VSCode (https://code.visualstudio.com/)
- IDE
    - PyCharm (https://www.jetbrains.com/pycharm/)

Tutorial

- The Canonical Tutorial from the creators:

https://docs.python.org/3/tutorial/index.html
(https://docs.python.org/3/tutorial/index.html)

- It's pretty long and a little verbose, but it is good, and the authoritative source.

Python’s Built-in Types

- numerics - `int`, `float`, `complex`
- sequences - `list`: `[ ]`, `tuple`: `()`, `range`, `str`: `''`, etc.
- mappings and sets - `dict`: `{}`, `set`, `frozenset`
- Others - iterators, generators, binary sequences, memoryviews, classes, instances, exceptions, modules

Each type has infix operators (like `*` and +) and methods (like `.hex()` or `.center()`). that work on them.

Numerics - `int`, `float` and `complex`

https://docs.python.org/3/library/stdtypes.html#typesnumeric
(https://docs.python.org/3/library/stdtypes.html#typesnumeric)

```
In [1]: 1 + 1
Out[1]: 2
```

```
In [2]: 1 + 1.0
Out[2]: 2.0
```

Python will implicitly cast a result to `float` if you include a decimal.

Sequences - **`list`**, **`tuple`**, and **`string`**

https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range (https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range)

In [3]:

```
l = [1, 2, 3, 4, 5]
t = (1, 2, 3, 4, 5)
s = '12345'
```

Sequences can be indexed ..

```
In [4]: l[0], t[0], s[0]

Out[4]: (1, 1, '1')
```

Sequences can be sliced ...

```
In [5]: l[1:3], t[1:3], s[1:3]

Out[5]: ([2, 3], (2, 3), '23')
```

Sequences can be tested for membership ...

```
In [6]: 1 in l, 1 in t, '1' in s
```

```
Out[6]: (True, True, True)
```

```
In [7]: 9 in l, 9 in t, '9' in s
```

```
Out[7]: (False, False, False)
```

Sequences can be looped over ...

In [8]:

```
for i in l:
    print(i)
```

```
1
2
3
4
5
```

What is the difference between the `list` and the `tuple`?

```
In [9]: l.append(6)
        l
```

```
Out[9]: [1, 2, 3, 4, 5, 6]
```

```
In [10]: # Doesn't work
         # t.append(5)
```

A `tuple` is immutable (cannot be changed), while a `list` is mutable.

Mappings - `dict`

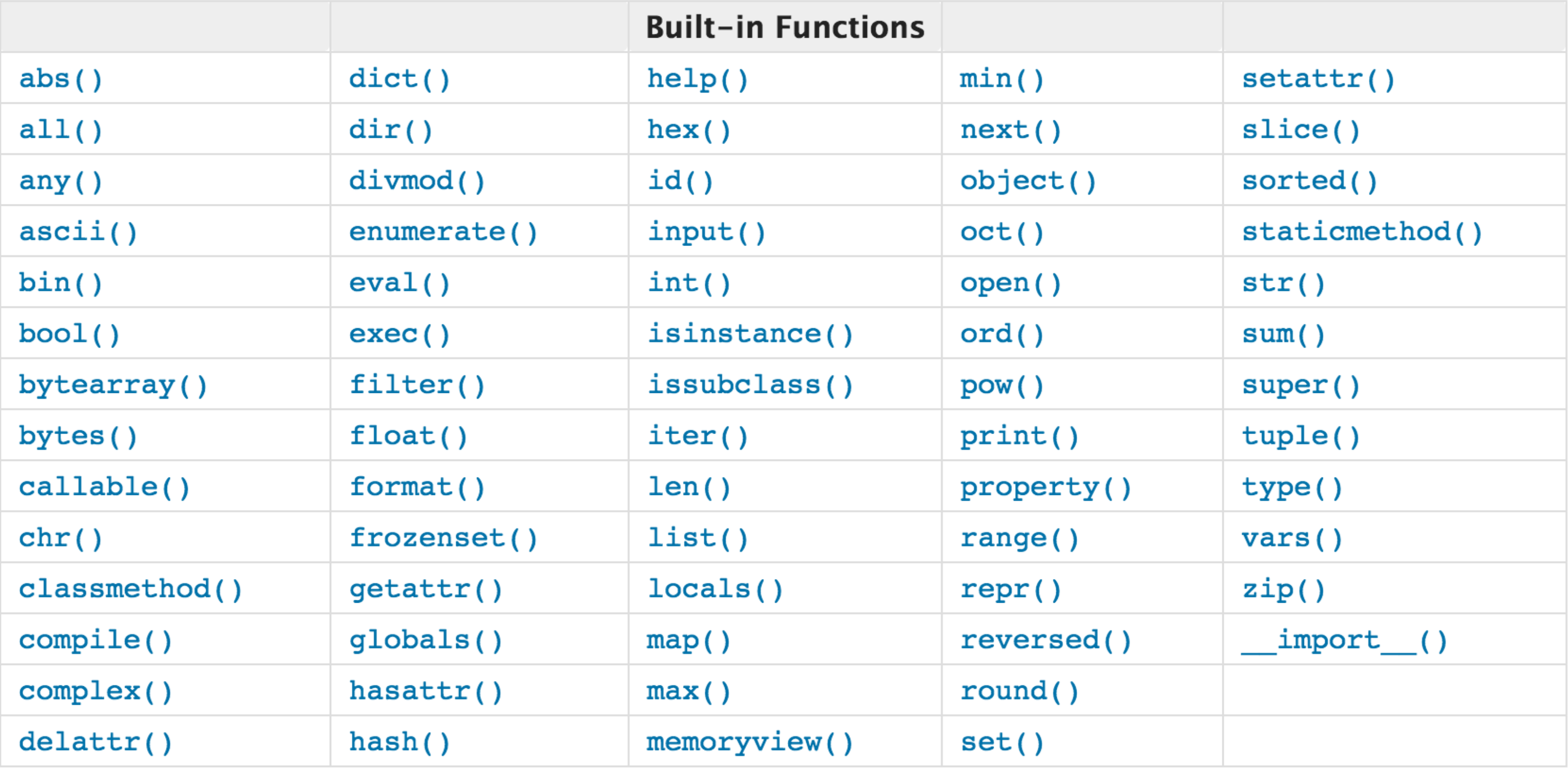
https://docs.python.org/3/library/stdtypes.html#mapping-types-dict
(https://docs.python.org/3/library/stdtypes.html#mapping-types-dict)

```
In [11]: d = {"name": "Austin"}
         d["name"]

Out[11]: 'Austin'
```

```
In [12]: d["height"] = "6 ft"
         d

Out[12]: {'name': 'Austin', 'height': '6 ft'}
```

Python Language Syntax

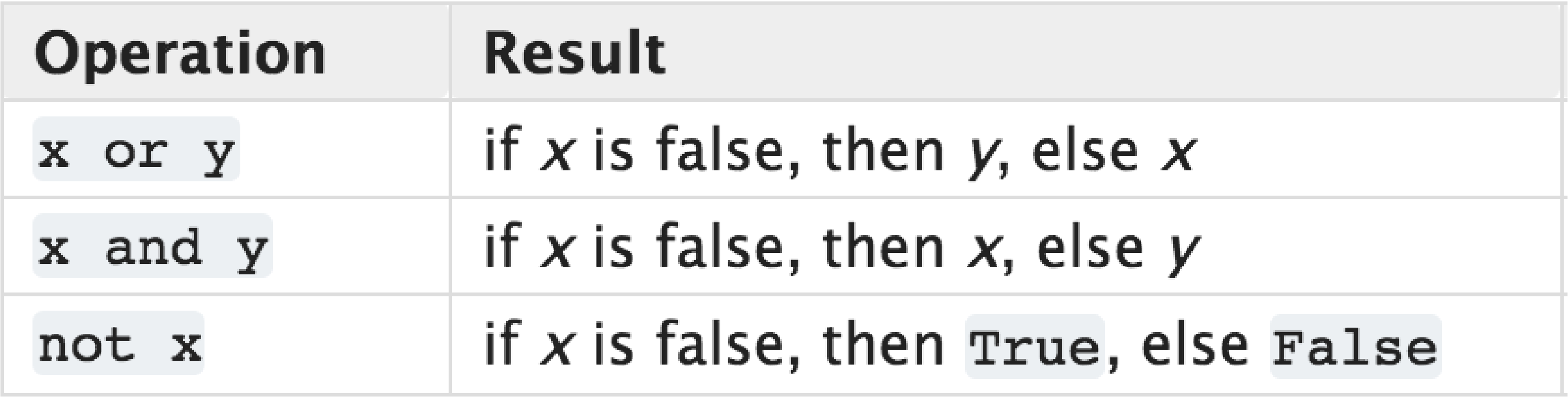
Built-in Functions



		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Boolean Operations

- `False` things: `False`, `None`, `0`, `0.0`, `''`, `()`, `[]`, `{}`, any object whose `__bool__` or `__len__` method returns a `False` value.
- Everything else is true.

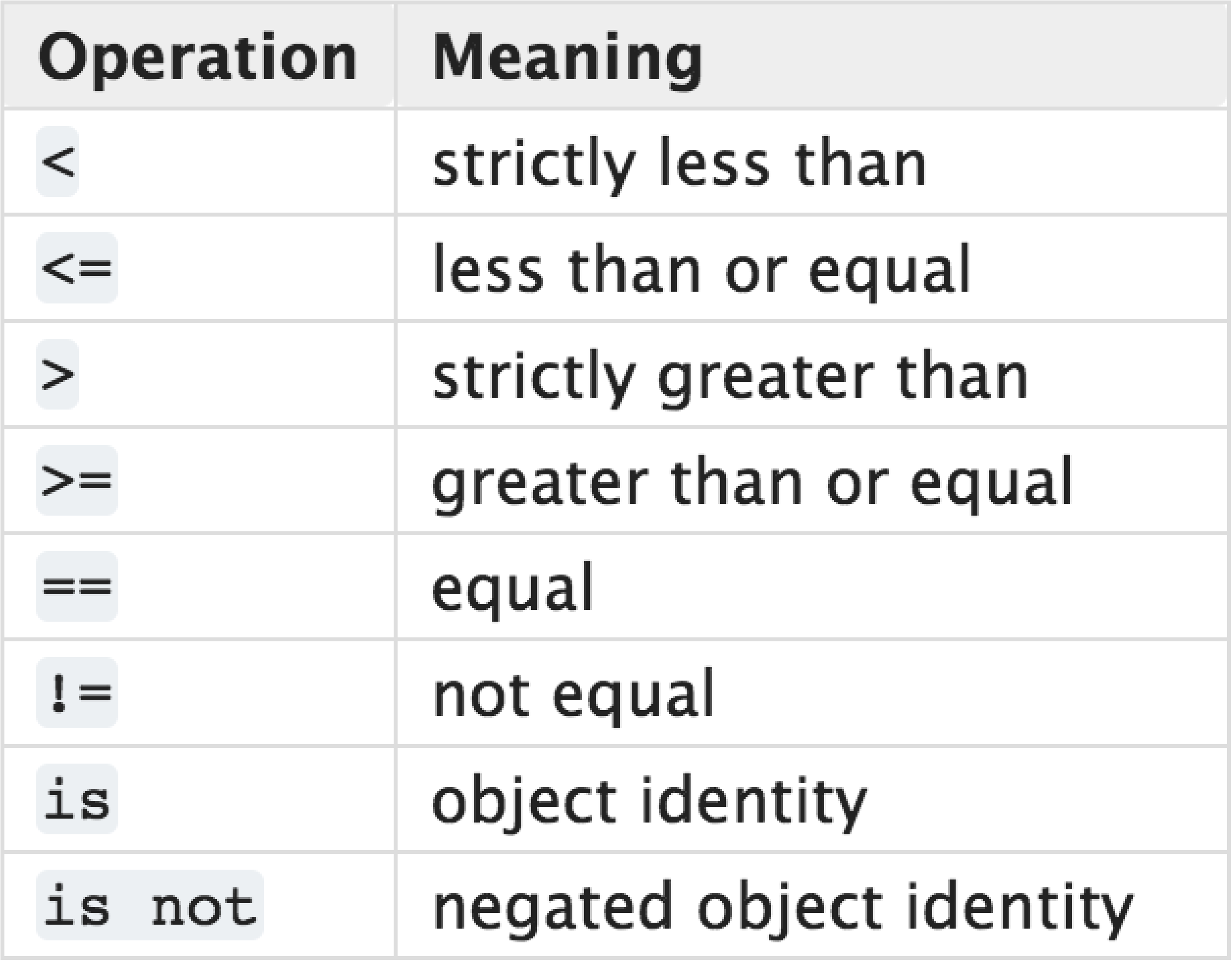


Operation	Result
`x or y`	if *x* is false, then *y*, else *x*
`x and y`	if *x* is false, then *x*, else *y*
`not x`	if *x* is false, then `True`, else `False`

Built-in Constants

- `False`, `True`
- `None`
- `NotImplemented`
- `Elipsis` (same as `...`)
- `__debug__`
- `quit()`, `exit()`, `copyright`, `license`, `credits`

Comparisons



Operation	Meaning
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal
`is`	object identity
`is not`	negated object identity

Control Structures

if, else, elif, for, while, break, continue

In [13]:

```
x = [1, 2, 3]
for i in x:
    print(i)
```

```
1
2
3
```

Data and Execution Model

To really understand the guts of Python, after you get the general syntax understood, read the Data Model and Execution Model docs:

- https://docs.python.org/3/reference/datamodel.html (https://docs.python.org/3/reference/datamodel.html)
- https://docs.python.org/3/reference/executionmodel.html (https://docs.python.org/3/reference/executionmodel.html)

Examples

Let's look at some examples:

```
In [14]: print("Hello, World!")
```

```
Hello, World!
```

A function

In [15]:

```
def hello1():
    print("Hello, World!!!!")

hello1()
```

```
Hello, World!!!!
```

A function with a keyword argument

In [16]:

```
def hello2(name=None):
    if not name:
        name = "World"
    print(f"Hello, {name}")
    print("Hello, " + name + "!")

hello2()
hello2("Skippy")
```

```
Hello, World
Hello, World!
Hello, Skippy
Hello, Skippy!
```

Object Oriented Programming and Python

Python is an object oriented programming language, but it doesn't for YOU to write your code that way. You can write procedural or semi-functional code. Doing so is very common.

Everything in Python is an object.

Classes

Python's simplest class as an example of the dynamic nature of Python.

In [17]:

```
class Classy:
    pass

c = Classy()
c.foo = 'Lobsters!'
c.bar = lambda x: x**2

print(c.foo, c.bar(3))
```

```
Lobsters! 9
```

In [18]:

```
class Person1:
    """Class representing a person, for providing Greetings."""

    def __init__(self, name):
        self.name = name

    def greet(self):
        print(f"Hello, {self.name}")

skippy = Person1("Skippy")
skippy.greet()
print(skippy.name)
print(skippy)
```

```
Hello, Skippy
Skippy
<__main__.Person1 object at 0x111b90b00>
```

In [19]:

```
class Person2:
    """Class representing a person, with greetings and height."""

    def __init__(self, name, height):
        self.name = name
        self.height = height  # Height of person in inches

    def greet(self):
        print(f"Hello, {self.name}")

    @property
    def height_ft(self):
        return self.height / 12.0

chip = Person2('Chip', 70)
print(chip.height)
print(chip.height_ft)
print("Chip is %.2f tall" % chip.height_ft)
```

```
70
5.833333333333333
Chip is 5.83 tall
```

Inheritance

In [20]:

```
class Ninja(Person2):
    """A stealthy person or 1337 hacker."""
    ninja_types = ["stealth", "hacker"]

    def __init__(self, name, height, ninja_type):
        super().__init__(name, height)
        if ninja_type in self.ninja_types:
            self.ninja_type = ninja_type
        else:
            raise RuntimeError('Invalid ninja_type: %s' % ninja_type)

    def work(self):
        if self.ninja_type == 'stealth':
            print("Karate Chop!")
        elif self.ninja_type == 'hacker':
            print("Hack hack hack.")
```

In [21]:

```
wally = Ninja('Wally', 62, 'hacker')
print("{name} is {height:.2f} tall.".format(name=wally.name, height=wally.height_f
t))
wally.work()
```

```
Wally is 5.17 tall.
Hack hack hack.
```

Exceptions

You see that `raise` in the class definition for `Ninja`?

```
In [22]: try:
             webster = Ninja('Webster', 71, 'quilting')
         except RuntimeError as e:
             print('No such thing as a Quilting ninja! \nError: %s' % e)
         finally:
             print('Nice work!')
```

```
No such thing as a Quilting ninja!
Error: Invalid ninja_type: quilting
Nice work!
```

What’s up with the ”””?

It's called a `docstring`, you can use them on modules, functions and classes. There's a whole ecosystem of tools designed to use them for documentation and testing. Use 'em!

```
In [23]:  Ninja.__doc__

Out[23]:  'A stealthy person or 1337 hacker.'
```

More OOP?

If OOP is a good fit for your problem, I've found this to be a great post on OOP in Python:

https://jeffknupp.com/blog/2014/06/18/improve-your-python-python-classes-and-object-oriented-programming/ (https://jeffknupp.com/blog/2014/06/18/improve-your-python-python-classes-and-object-oriented-programming/)

Context Managers

"A context manager is an object that defines the runtime context to be established when executing a with statement."

Turns this ...

```
In [24]: f = open('file.txt', 'r')
         print(f.read())
         f.close()
```

```
I'm a text file!
```

into this ...

```
In [25]: with open('file.txt', 'r') as f:
             print(f.read())
```

```
I'm a text file!
```

The Standard Libraries

Lots of fabulous tools we don't have time for, things like:

- fancy data types like datetimes and calendars
- path and file manipulation
- basic math
- Logging, curses, network protocols ... on and on

Dive in! https://docs.python.org/3/library/index.html (https://docs.python.org/3/library/index.html)

In [26]:

```
import math

a = 3.5
math.ceil(a), math.floor(a), math.pi
```

Out[26]:

```
(4, 3, 3.141592653589793)
```

Last warning on Python 3 vs 2.

If you have to write portable code, read up on it, it's messy but not too bad. Lots of people have managed it, you can too! Look for the package `six`.

3rd Party Libraries

Stick around for Intro to Python Part II

3rd party libraries combined with the language's low barrier to entry are Python's competative advantage.

Thank You!

Austin Godber
@godber
DesertPy Co-Organizer
Meetup, Github and http://desertpy.com (http://desertpy.com)

- https://github.com/desertpy/presentations (https://github.com/desertpy/presentations)
- godber-intro-to-python
- godber-intro-to-python-part-II

Desert Code Camp 2019 - 10/12/2019